Bike Sharing Demand

Prepared by: Soumee Ghosh (2211444)

Data Description:

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. In this competition, participants are asked to combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

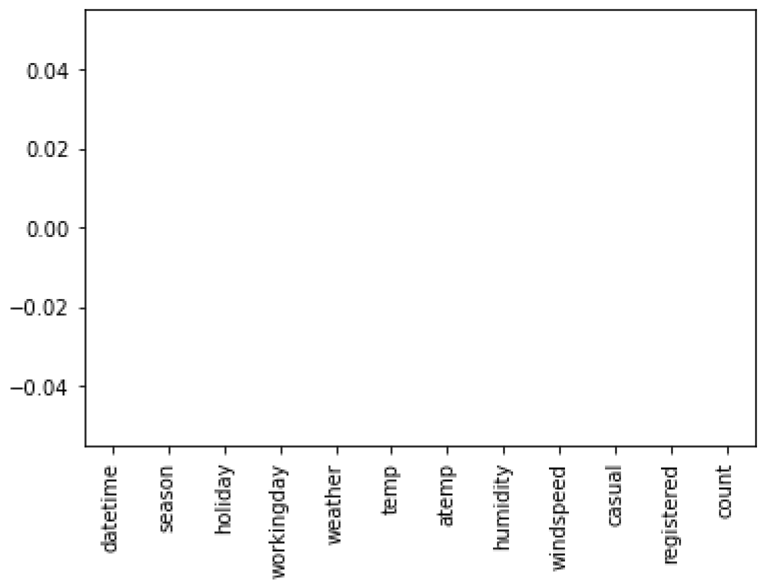
The analysis can be performed in the following way:

1. Loading and Understanding the data:

   At first we load the training and testing data and parse datetime column. Using the .info() method on the training data we can observe that there are 10886 entries and 12 features. Thereafter we check the statistical description of the training data using .describe() method

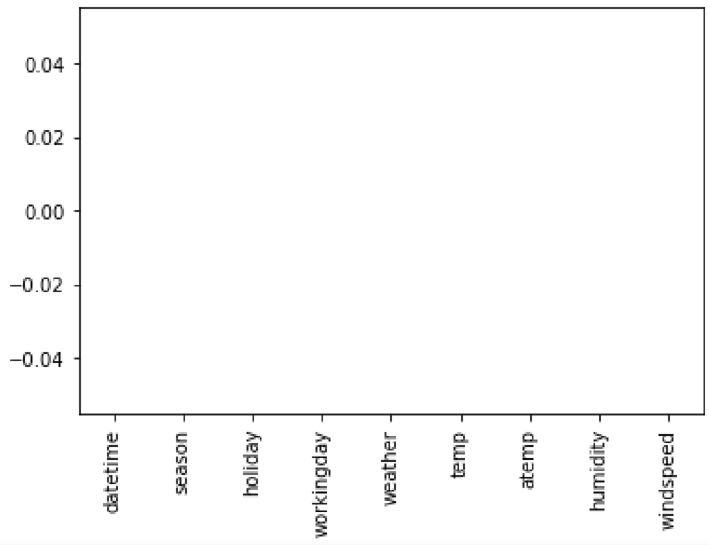
   Using the .isna() method we check for presence of null values in the dataset.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3db56e50>
```



1. no null values present in the training dataset

Similarly we check for presence of null values in the test set.



1. No null values present in the test set

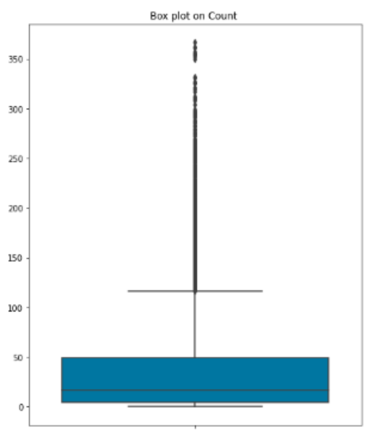
Then we extract date, time, day, month, week , etc from the datetime feature of the dataset using the following function and pandas.

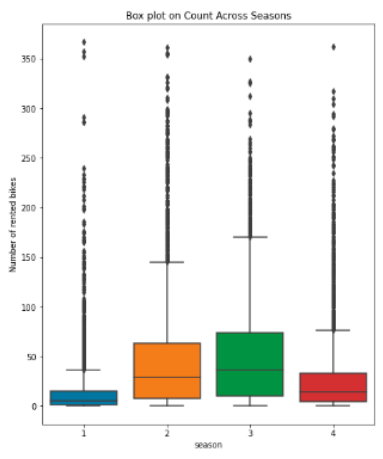
```
]: def convert_date(df):


       df['datetime'] = pd.to_datetime(df['datetime'])
       df['month'] = df['datetime'].dt.month
       df['hour'] = df['datetime'].dt.hour
       df['weekday'] = df['datetime'].dt.dayofweek
       df["day"]=df["datetime"].dt.day
       df["year"]=df["datetime"].dt.year
       df['dayofweek'] = df['datetime'].dt.dayofweek
       df['month_start'] = df['datetime'].dt.is_month_start


       return df
```

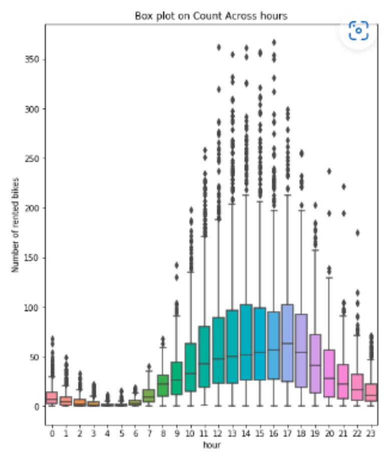
```
]: data=convert_date(train)
```

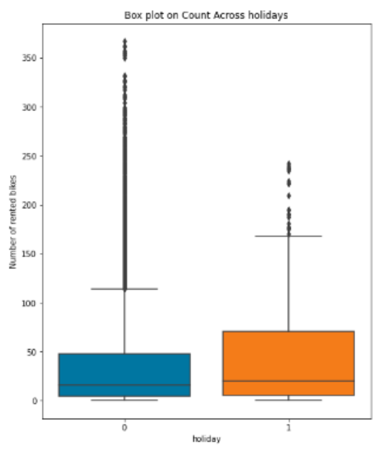
2. EDA

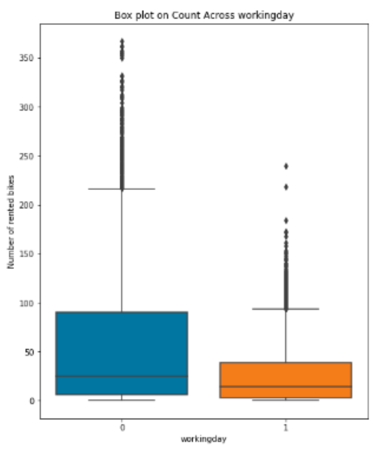
Then we perform some EDA

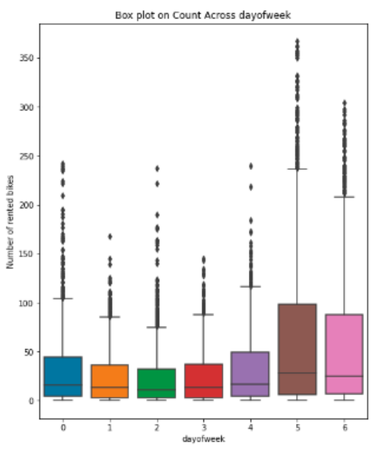






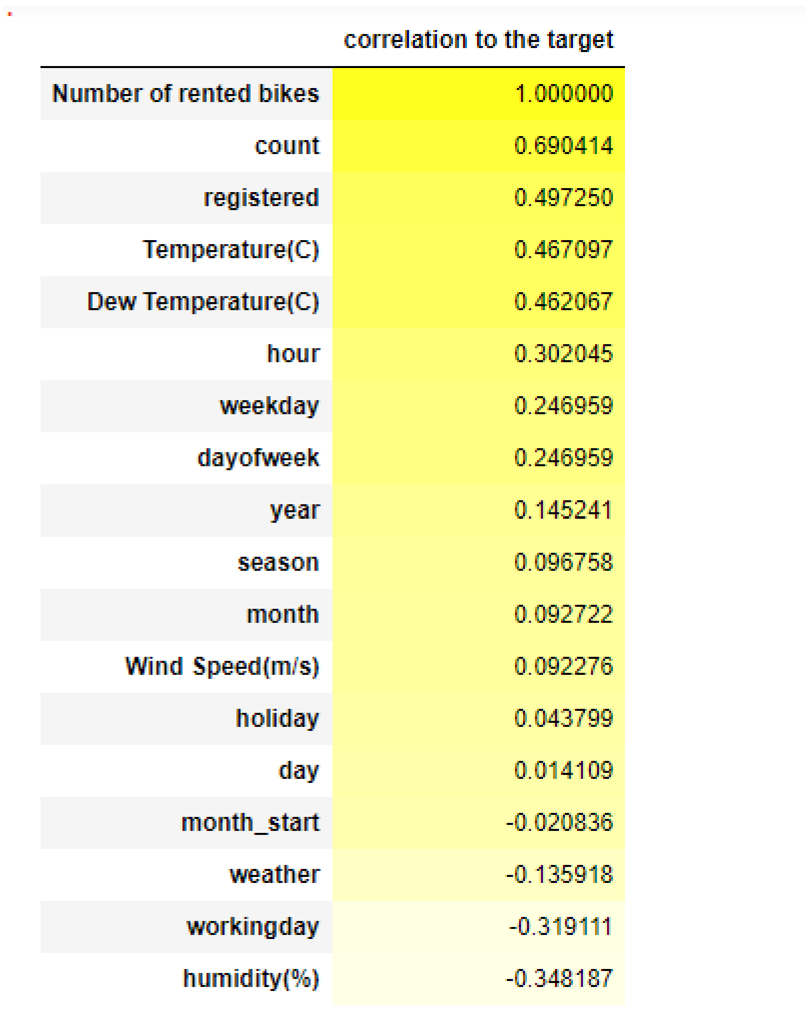






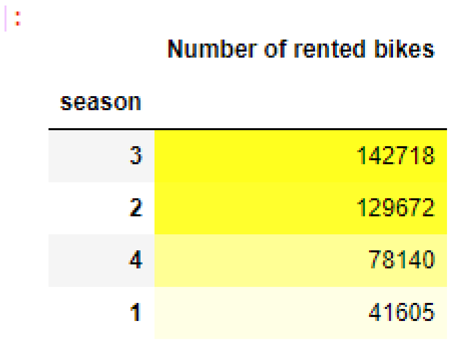
We try to find the count of the number of bikes rented across Seasons, holidays, hours, working day and days of the week.
We observe that during the season 2 and 3 most number of bikes are rented. For hours , we observe that most number of bikes are rented during the daytime and it falls as we proceed the night time.
For the day of the week we find that most number of bikes are rented on the 5th and 6th day with o being Monday so most bikes are rented during Saturday and Sundays. So we can also conclude from here that more bikes are rented on weekends compared to working days.

Next we perform the correlation and observe the relation between the target and the features.



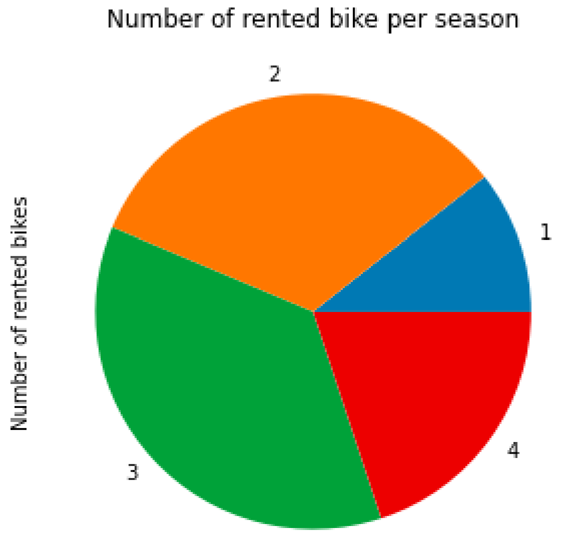
	correlation to the target
Number of rented bikes	1.000000
count	0.690414
registered	0.497250
Temperature(C)	0.467097
Dew Temperature(C)	0.462067
hour	0.302045
weekday	0.246959
dayofweek	0.246959
year	0.145241
season	0.096758
month	0.092722
Wind Speed(m/s)	0.092276
holiday	0.043799
day	0.014109
month_start	-0.020836
weather	-0.135918
workingday	-0.319111
humidity(%)	-0.348187

We then find the number of bikes rented across seasons.



	Number of rented bikes
season	
3	142718
2	129672
4	78140
1	41605

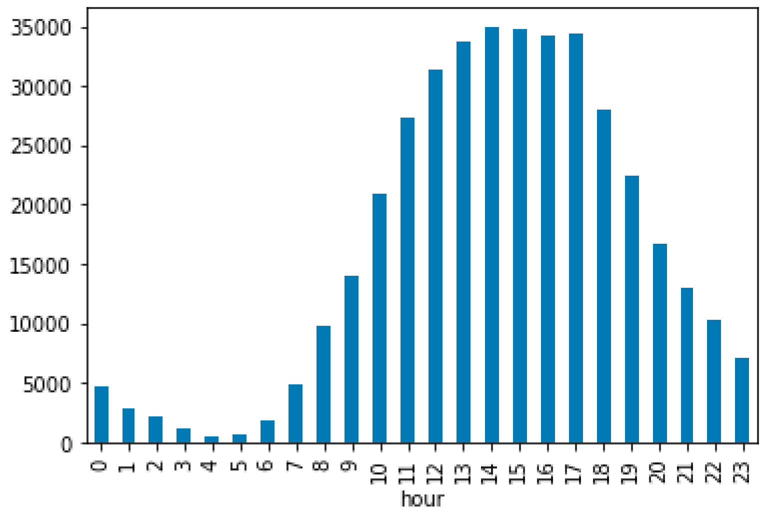
```
Text(0.5, 1.0, 'Number of rented bike per season')
```



We then find the number of bikes rented by hour.

```
data.groupby('hour').sum()['Number of rented bikes'].plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa41d9bac0>
```



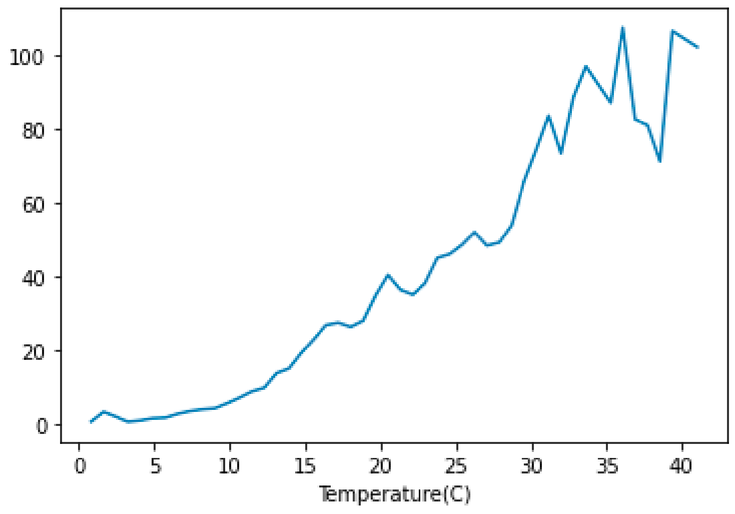
```
34]:
```

dayofweek	Number of rented bikes
5	100782
6	90084
4	47402
0	46288
3	37283
1	35365
2	34931

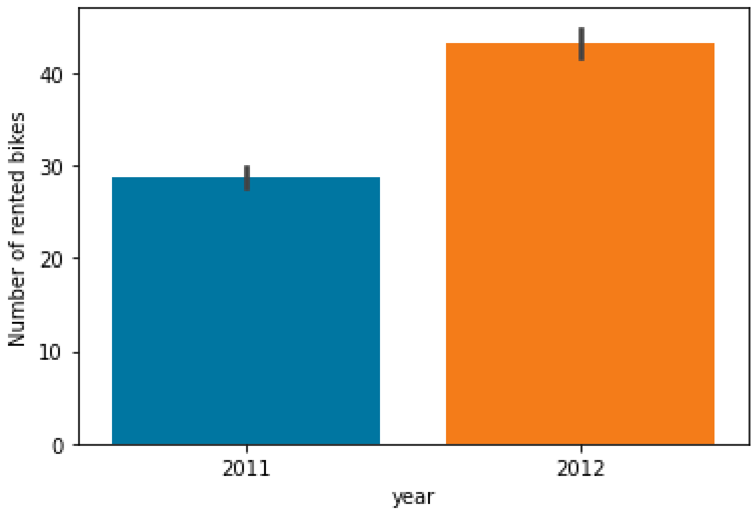
We observe that the number of bikes rented increases with increase in temperature.

```
data.groupby('Temperature(C)').mean()['Number of rented bikes'].plot.line()
```

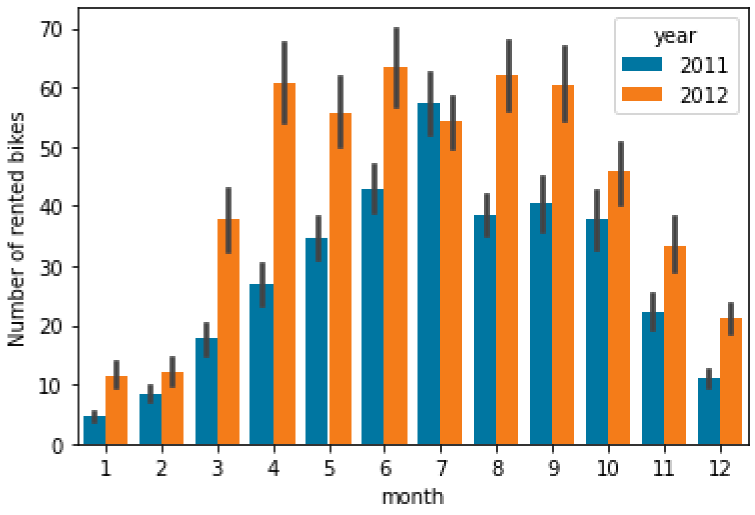
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa413ff070>
```



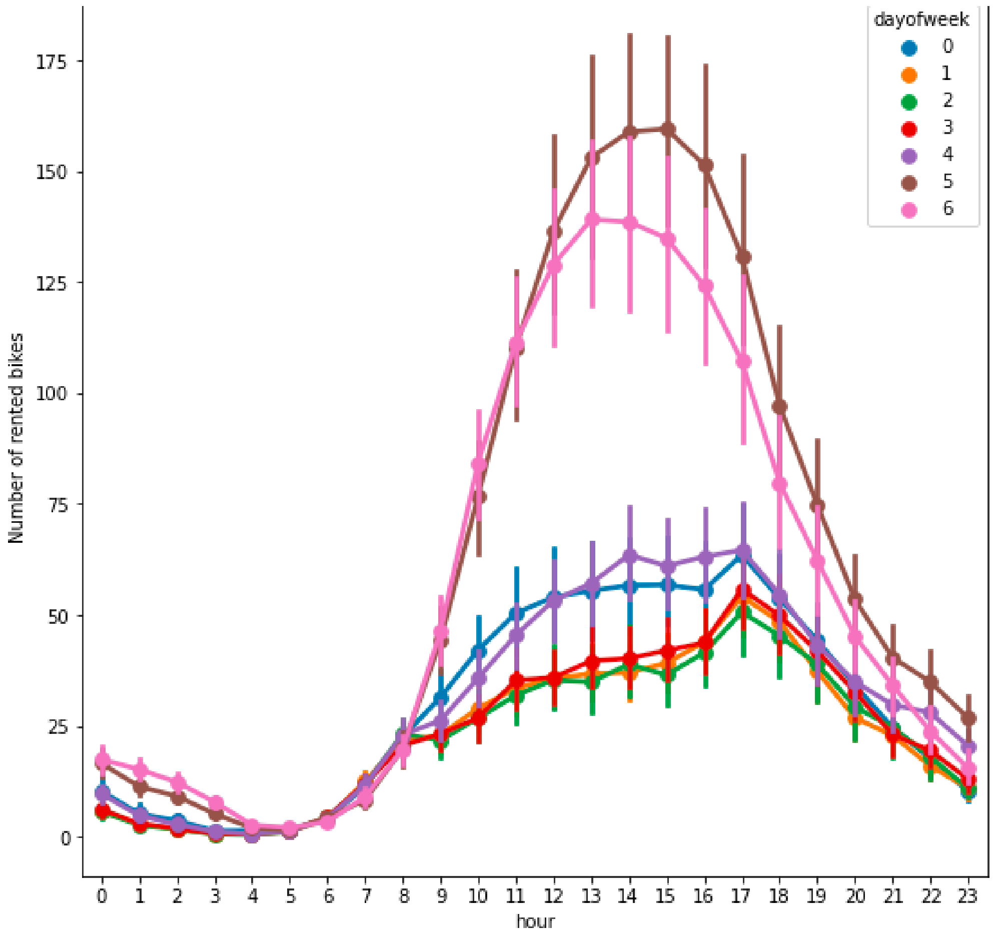
We observe that the number of bikes rented in 2012 is more compared to 2011. So more people are renting bikes, the demand is increasing.



```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa414d2bb0>
```

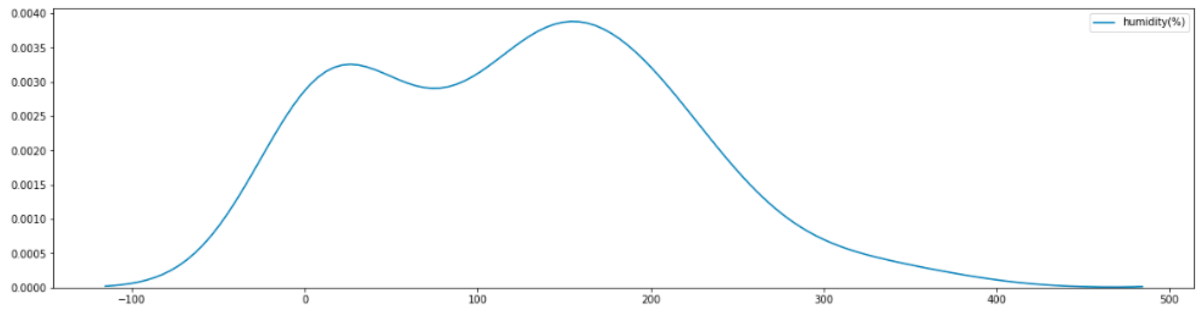


Here we can observe that for every month the demand for renting bikes is increasing n 2012 compared to 2011 , only exception is the 7th month.



This shows the number of bikes rented per hour for each day of the week. So we can observe similarity in the hours for every day with more bikes rented on 5th and 6th day.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa414608b0>
```



This shows the demand in rented bikes with respect to humidity.

3. Model building and Data Pre Processing

Here we scale the data and split the data into feature and target variable. Thereafter we split the data into training and validation set and perform the necessary model building.

We train our model using the following models and observe their rmse scores.

```
Training model: LinearRegression
root mean squared error for  LinearRegression : 147.13815467083367
Training model: RandomForestRegressor
root mean squared error for  RandomForestRegressor : 66.14350821522432
Training model: XGBRegressor
root mean squared error for  XGBRegressor : 65.02987529284
```

Since XG boost performs the best so we tune the hyperparameters of the XG boost model using Randomized CV.

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
▸    RandomizedSearchCV
▸ estimator: XGBRegressor
      ▸ XGBRegressor
```

Then using the best hyperparameter values we train the model.

```
# Printing the best score that we can get using the parameters
print('We can get score of :',random_search.best_score_,' using',random_search.best_params_)
```

```
We can get score of : 0.9099637493068711  using {'subsample': 0.8, 'n_estimators': 750, 'min_child_weight': 10, 'max_depth': 1
0, 'learning_rate': 0.1, 'gamma': 0.5, 'colsample_bytree': 0.8}
```

```
# Creating XGBRegression model with the select hyperparameters
xgb_reg_hpt = XGBRegressor(subsample=0.8, n_estimators=750, min_child_weight=10,max_depth=10,
                              learning_rate=0.1, gamma=0.5, colsample_bytree=0.8, n_jobs = -1)
```

```
xgb_reg_hpt.fit(X_train,y_train)
```

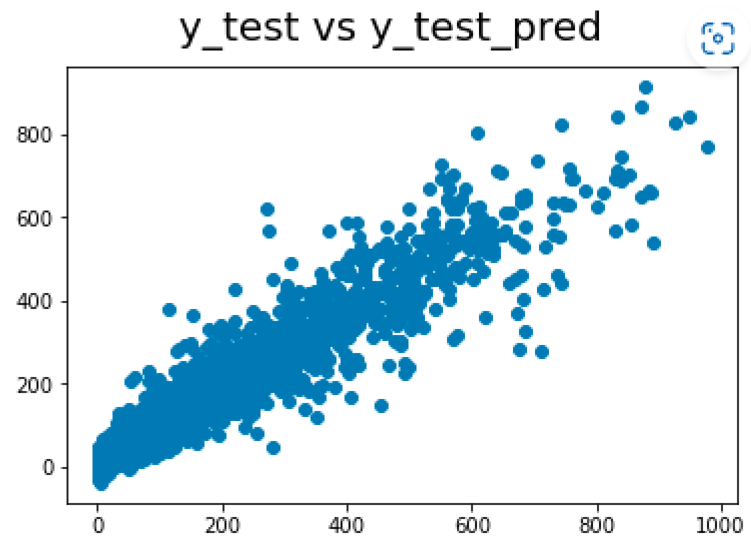
```
▾                          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.8,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0.5, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
```

We observe that the rmse value improves to 60.73 due to hyperparameter tuning

```
R2 score on Training data :  100.0
R2 score on Test data :  89.0
```

We then observe the R2 score.

```
: <matplotlib.collections.PathCollection at 0x1fa3ff3c250>
```


y_test vs y_test_pred
800
600
400
200
0
0
200
400
600
800
1000